(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 April 2002 (04.04.2002)

PCT

(10) International Publication Number
WO 02/27443 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US01/22624

(22) International Filing Date: 24 August 2001 (24.08.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/668,338 25 September 2000 (25.09.2000) US

(71) Applicant *(for all designated States except US)*: **ITT MANUFACTURING ENTERPRISES, INC.** [US/US]; 1105 North Market Street, Wilmington, DE 19801 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **ZALESKI, Philip, J.** [US/US]; 775 Daedalian Drive, Rome, NY 13441 (US). **VIENNEAU, Robert, L.** [US/US]; 775 Daedalian Drive, Rome, NY 13441 (US).

(74) Agents: **FINNAN, Patrick, J.** et al.; Epstein, Edell, Shapiro, Finnan & Lytle, LLC, Suite 400, 1901 Research Boulevard, Rockville, MD 20850 (US).

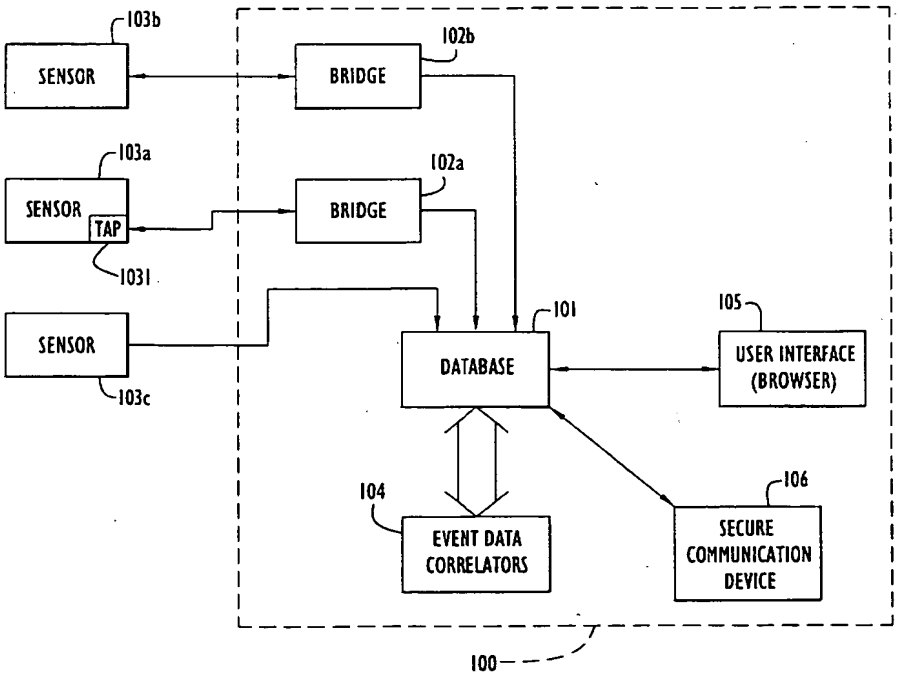
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: GLOBAL COMPUTER NETWORK INTRUSION DETECTION SYSTEM



(57) Abstract: A method and system for enabling detection of concerted intrusion efforts by collecting data from various local sensors resident in intrusion detection system log files, firewalls, and routers across multiple network sites and normalizing, correlating and processing ("integrating") the sensor data from multiple different sensors resident on the multiple network sites.

WO 02/27443 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

GLOBAL COMPUTER NETWORK INTRUSION DETECTION SYSTEM

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to computer network security and detection of attempted accesses into or attacks on secure computer networks by unauthorized persons or unauthorized computers. More particularly, the present invention relates to a system and method for detecting attempted intrusions into a secure computer network or networks by receiving and processing network sensor information from multiple sensors, each detecting the occurrence of a particular event. The system receives and integrates at a central location the information from all of the sensors and processes the integrated information according to predefined relationships among the sensor outputs, so as to more accurately detect an attack upon a secure network, while minimizing the occurrence of false alerts. The system is particularly suited for geographically distributed and interconnected local area networks (LANs), or wide area networks (WANs), and enables detection of coordinated attacks across different levels of the network, such as at the regional level.

Background and Related Art

It is generally known in the art to monitor network events for the purpose of detecting an attempt to intrude or break into the network, either to gain unauthorized access to databases, programs, and files on network servers, or rather than focusing on a particular weakness of the computing system, to debilitate it by bombarding it with offensives from so many angles that normal computation is simply impossible. This type of intrusion constitutes a denial of service attack.

In responding to an attack, two different and opposing forces are present. On the one hand, it would be desirable to increase the pressure of a defense in order to fight off more and more of the attack. At the same time, however, that increase would limit the computational resources (processor cycles, disk space, I/O usage) available for legitimate work. It is the job of the response system to balance these two needs in order to maintain a safe, usable system.

The Common Intrusion Detection Framework (CIDF) is an effort among DARPA (Defense Advanced Research Projects Agency)-funded intrusion detection projects to define a standard interface for intrusion detection and response components. The objective is to allow independently designed components to share information without requiring extensive translation with each new integration. The CIDF is explained in Staniford-Chen, S., et al, The Common Intrusion Detection Framework (CIDF). Position paper accepted to the Information Survivability Workshop, Orlando FL, October 1998. The CIDF Matchmaking Service, or matchmaker, provides a standard, unified mechanism for CIDF components to make themselves known to other

components, and for components to locate communication "partners" with which they can share information. A single infrastructure is used for this purpose to promote component interoperation and ease development of multi-component intrusion detection and response systems.

While computer network intrusion detection systems thus are known, there remains in the art a need to be able to detect in near real time concerted intrusion attempts across multiple network sites. Such enhanced intrusion detection capability has not been possible in the prior art systems. In particular, such enhanced intrusion detection cannot be realized in the prior art systems because of the scattered distribution of activity sensors at multiple local servers, and the fact that the data produced by multiple sensors from many different manufacturers is based on many diverse formats which are incompatible with each other.

SUMMARY OF THE INVENTION

The present invention provides a system which enables detection of concerted intrusion efforts by collecting data from various local sensors resident in intrusion detection system log files, firewalls, and routers across multiple network sites and normalizing, correlating and processing ("integrating") the sensor data from multiple different sensors resident on the multiple network sites.

In particular, the present invention provides a system for detecting attempted intrusions into a computer network composed of a plurality of computer hosts, including a central intrusion detection database system, a tap residing on at least one computer host, which reads event data from activity log files created by an event

sensor provided on the host and sends the event data to said central database system, the central database system including for each tap a bridge which receives and processes the sensor information from the tap, and stores the processed sensor information in a database, at least one correlator which receives the processed sensor information from the database, processes the information according to a predefined algorithm to generate a correlated event, and stores the correlated event in the database, and a user interface for receiving the information stored in the database and presenting the received information to a user for monitoring and evaluation.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described in detail with reference to the following drawings in which:

Fig. 1 is a block diagram of a computer network intrusion detection system according to one preferred embodiment of the present invention;

Fig. 2 is a schema diagram for operational flow of a sensor tap (1031 of Fig. 1);

Fig. 3 is a schema diagram for operational flow of a bridge (102a of Fig. 1);

Fig. 4 is a block diagram showing the relationship of the bridge and sensor with respect to a boundary device;

Fig. 5 is a flow chart diagram of the operation of the sensor tap according to one embodiment of the invention;

Fig. 6 is a flow chart diagram of the operation of the bridge according to one embodiment of the invention;

Fig. 7 is a block diagram showing different event data correlator modules according to one embodiment of the invention;

Fig. 8 is a schematic diagram illustrating the types of records stored in the database according to one embodiment of the present invention; and

Figs. 9A, 9B-1, 9B-2, and 9C are flow chart diagrams of the operation of the category and IP correlators according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Fig. 1 shows an example of the global computer network intrusion detection system according to one preferred embodiment of the invention. The global intrusion detection system (GIDS) 100 uses data from existing intrusion detection systems, firewall data files and router data files to collect event data into a database 101 for integration and analysis. Database 101 is preferably an Oracle® database, but other suitable commercially available relational databases also could be used in the invention. The system 100 includes a number of event data correlators 104, provided in the form of separate correlator modules running in parallel. Each correlator module receives new event data from the database 101 and processes it according to a specific procedure in order to determine whether a correlated event has occurred, which would indicate the likelihood of a concerted intrusion attack. The correlators 104 store correlated event information back to the database 101.

System 100 further includes a user interface 105, preferably in the form of a graphical browser similar to the type used to view html documents; a secure

communication device 106, for communicating and interfacing with other servers and/or clients on the computer network being monitored; and a number of bridges 102a and 102b. Bridges 102 establish TCP (transfer communication protocol) sockets with sensors 103a, 103b, and collect event data from the sensors for entry into the database.

Sensors 103 can be sensors specifically manufactured for operation with the GIDS 100 of the invention, or can be any of a number of commercially available sensors for routers, firewalls or servers. The sensors themselves can be composed of consoles or managers and distributed data agents or collectors. Examples of such sensors include ASIM, JIDS, ISS RealSecure, Cisco Router, Sidewinder Firewall, Raptor Firewall, Gauntlet Firewall, NetRanger, and NetRadar. Data feeds from sensors 103 are pre-processed in a near real-time manner and stored in the database 101 for further processing by the event correlators, and display by the user interface.

Many commercial off the shelf (COTS) or government off the shelf (GOTS) sensor tools 103a write event data to a local log file where the sensor is resident. To collect and integrate this log file data, the present invention provides a piece of customized software code called a sensor tap 1031 and places it on the local server hosting the sensor 103a. The sensor tap 1031 pre-processes the event data stored in the log file by the associated sensor and transmits it to the GIDS system 100.

Other types of sensors 103b provide a server process with which the GIDS system can communicate. For such sensors a stand-alone bridge 102b is provided. Bridge 102b connects to a specific port on the sensor system 103b

and reads each event as it is made available by the sensor system. Accordingly, no sensor tap is needed for this type of sensor.

It is also possible for a sensor system 103c to be specially constructed which includes an interface that can communicate directly with the database 101. In this case event transfer and insert rates into the database would be faster than using the bridge/tap interface since it would not need to be parsed, encrypted, decrypted or inserted into the database by any specialized code. Because the COTS and GOTS sensors are the most prevalent, a detailed description will be given of the bridge/tap interface used for such sensors.

Referring to Fig. 2, the schema of the sensor tap according to one embodiment of the present invention is shown. At step 201, the tap retrieves configuration information stored in a local text file called "CONFIG_FILE." The CONFIG_FILE includes the following information: **IP**, or the IP address of the system on which the bridge associated with the sensor tap resides (e.g., IP 255.255.255.255); **COMM_PORT**, or the local port on which the tap sets up a TCP server for communication with the bridge (e.g. COMM_PORT 44444); **LOG_DIR**, or the full path to the sensor's log file (e.g., LOG_DIR /user/local/log/); **LOG_FILE**, or the name or wildcard of the name of the sensor's log file (e.g., LOG_FILE logfile, or attacks.log); and **ENCRYPTION**, a designator indicating whether data encryption is to be turned on or off (e.g., ENCRYPTION on).

Once the configuration information has been retrieved, the sensor at step 202 builds a TCP server for connection to the specified IP, which listens for the

specified IP (i.e., waits for a signal from the bridge containing the IP address) to connect to the TCP server on the specified port. The TCP server continues to listen until the specified bridge IP has been detected at step 203. At step 204 the TCP server accepts the request for connection from the bridge, if the IP of the requester matches the IP specified in the CONFIG_FILE, and establishes a TCP socket between the two hosts. If the requester's IP does not match the specified IP stored in the CONFIG_FILE, the TCP server serves an error message to the requester and exits.

At step 205 the tap scans the sensor system to locate the active log file pointed to by the log file in the CONFIG_FILE and to open the file. At step 206 the active log file is scanned by the tap to be read for any new events that have become available, and stores the new events in a character string. At step 207, the event character string is specifically parsed into a number of parsing categories, as follows: **date_time**: the date and time of the event in the format DD-Mon-YYYY HH:MM:SS); **sensor_name**: the name of the sensor system from which the data is being read (e.g., JIDS, RAPTOR); **source_ip**: the IP address of the attacking host (e.g., 123.13.12.213); **dest_ip**: the IP address of the target host (e.g., 213.31.21.123); **description**: a description of the event, which can include various pieces of data; **source_name**: the fully qualified domain name of the attacking host, if applicable (e.g., attacker.whynot.com); **dest_name**:the fully qualified domain name of the attacking host, if applicable (e.g., sitting.duck.com); **signature**: the unique event classifier which is provided by the sensor system

(e.g., NMAP attack); **src_id**: the source ID which the sensor attaches to each event, if applicable; **source_port**: the communication port of the attacking host which was used during the attack (e.g.,2345); **dest_port**: the communication port of the target host which was used during the attack (e.g., 25); and **protocol**: the protocol which was used by the attacker during the event (e.g., FTP).

At step 208, an output string in a predefined format is created containing the parsed information. At step 209 the output string is encrypted for security if the CONFIG_FILE indicates that encryption is to be used. At step 210 the tap sends the character string in the form of a data stream to the bridge over the TCP socket. At step 211 the tap sends a heartbeat signal to the bridge over the TCP socket. The heartbeat signal is sent periodically, such as every one minute, in order for the bridge to confirm that the sensor tap is in a functioning state in the absence of any new event data.

Fig. 3 shows the schema of the bridge operational flow according to one embodiment of the present invention.

At step 301, the bridge obtains configuration information which is in the form of command line switches or flags in command lines for execution of the bridge processes. These switches or flags can be modified to alter the manner in which the bridge functions. Examples of the command line switches/flags are as follows:

-h {IP of the sensor system where the tap resides}
-p {communication port of the sensor system on which the tap TCP server is listening}
-l {location where the sensor resides} (e.g., BLDG1)
-s {name of the sensor} (e.g., JIDS}

-c {encryption on/off}
An example of a configuration command line argument for initializing a bridge is as follows:

/standard_bridge -h 255.255.255.255 -p 44444 -l BLDG1 -s JIDS -c on

At step 302 the bridge creates a TCP client for establishing communication with the TCP server of the tap. At step 303, the TCP client attempts to connect to the corresponding IP and port of the tap TCP server as specified in the configuration command line. At step 304, the bridge receives an accept acknowledgment message from the TCP server of the tap indicating that the TCP server accepted the bridge's request for connection. This establishes the TCP socket between the server and the client.

The bridge at step 305 receives an encrypted (if encryption was turned on) data stream over the TCP socket from the tap server containing the parsed event data. If encryption was specified, at step 306 the bridge decrypts the received data stream. At step 307, the bridge generically parses the decrypted data into specific variable categories, using a simple pipe-delimited parser. The parsing categories are as follows:

date_time: the date and time of the event in the format DD-Mon-YYYY HH:MM:SS);

sensor_name: the name of the sensor system from which the data is being read (e.g., JIDS, RAPTOR);

source_ip: the IP address of the attacking host (e.g., 123.13.12.213);

dest_ip: the IP address of the target host (e.g., 213.31.21.123);
description: a description of the event, which can include various pieces of data;
source_name: the fully qualified domain name of the attacking host, if applicable (e.g., attacker.whynot.com);
dest_name:the fully qualified domain name of the attacking host, if applicable (e.g., sitting.duck.com);
signature: the unique event classifier which is provided by the sensor system (e.g., NMAP attack);
src_id: the source ID which the sensor attaches to each event, if applicable;
source_port: the communication port of the attacking host which was used during the attack (e.g.,2345);
dest_port: the communication port of the target host which was used during the attack (e.g., 25); and
protocol: the protocol which was used by the attacker during the event (e.g., FTP).

At step 308, the bridge pre-filters the parsed data according to predefined filter criteria, such as certain source or destination IP address or port criteria, whereby if the event data matches the pre-filter criteria, it is not further processed. At step 309 the bridge sends the parsed event data to the database 101. At step 310 the bridge receives a heartbeat signal from the tap server. According to the preferred embodiment, the heartbeat signal is sent by the server over the TCP socket once per minute, to confirm that the tap is functioning normally. If the bridge does not receive the heartbeat signal, it will determine that there is a malfunction at the tap and take appropriate action (such as notifying management, setting an alarm, etc.).

An important aspect of the present invention pertains to the stateful inspection standard used for transmission of data between hosts. The stateful inspection standard allows external hosts to send data to hosts inside a protected boundary without having to open the boundary (which would subject the internal host to intrusion attacks). As shown in Fig. 4, a boundary device 401 (such as a firewall, a router, etc.) will allow incoming data traffic from an external host (such as sensor system 103) to an internal host (such as bridge system 102) over a TCP socket if the initial communication establishing the TCP socket came from inside the protected boundary.

This standard is used by the single integrated bridge/tap process by placing the tap TCP server outside the boundary, while the bridge TCP client is located inside the boundary (or "behind" the boundary device). As per the Berkeley TCP socket communication protocol, the bridge TCP client 403 initiates the communication (404) with the TCP server 402, which receives the communication request (405) through the boundary device 401. The server 402 accepts the communication message and sends an acknowledgment (406) to the client 403 through the boundary device. Because the acknowledgment communication from the TCP server 402 is in response to a communication initiated by the client 403 inside the boundary, the boundary device 401 allows the tap server to send data back through the boundary without opening a hole in the boundary. Since the boundary device keeps track of the destination IP address embedded in communications from the internal client, the boundary device can compare the source IP address embedded in incoming communications from outside the boundary with the destination IP addresses of

previously sent communications from clients inside the boundary. The TCP client 403 receives the acknowledgment of reception by the server (407), and the TCP socket (408, 409) is then established between the hosts through the boundary device, without opening any holes in the boundary.

The sequential operations of the sensor tap and the bridge will now be described with reference to Figs. 5 and 6. As shown in Fig. 5, at first a kick-start Daemon is initialized at step 501, to continually determine whether or not the tap is running (step 502). So long as the tap is running, the Daemon takes no action. If the tap is not running, the Daemon starts the tap at step 503. At step 504, the tap reads the config_file to obtain the configuration information, and at step 505 the tap creates the TCP server as explained above. The tap TCP server then enters a wait mode at step 506 for the bridge to connect to it by listening at the designated port for a communication signal from the bridge, at step 507. Once the connection attempt is detected, at step 508 the tap checks the identifying information in the connection message from the bridge and compares it with the configuration information obtained from the config_file to determine whether the requesting bridge is authorized to make the connection with the tap. If not, the tap causes the TCP server to terminate the connection at step 519. If the requester is authorized, a TCP socket is established between the server and the client as explained above, and at step 509 the tap opens the sensor's active log file and at step 510 scans to the end of the opened log file. At steps 511 and 512, the tap determines whether new data has been written to the log file, and

waits if no new data has been entered. The tap then proceeds to step 513 to read the new data and perform a specific parse of the read data. The parsed data is then formatted into an output character string at step 514. If the encryption feature is active as determined at step 515, the tap will encrypt the string at step 516. The output string is then transmitted to the bridge over the established TCP socket at step 517. The TCP server then determines whether the tap has exited at step 518. If so, the server exits at step 519. If not, the process returns to steps 511 and 512 to await the availability of new data placed into the active log file by the sensor.

As shown in Fig. 6, at step 601 the bridge is initialized, and at step 602 the bridge reads the configuration switches/flags passed to it in the initialization command lines as discussed above. At step 603 the bridge creates a TCP client according to the configuration settings it has read from the command lines. At step 604, the client attempts to connect to the tap TCP server, and waits for an accept acknowledgment at steps 605 and 607. If no acceptance is received within a predetermined timeout period in step 607, the client exits at step 618 and returns an error message. Once the connection has been established, at step 606 the bridge connects to the database 101 and at the loop of steps 608 and 609 the bridge waits for new data from the tap server. When new data is received, the bridge reads the event data at step 610 and determines whether or not decryption is needed at step 611. If the data is encrypted, the bridge decrypts it at step 612 and proceeds to step 613 to parse the data as discussed above. Otherwise, the unencrypted received data is directly parsed at step 613. At step 614

the bridge pre-filters the parsed data to determine whether the IP address or port matches the predefined filter criteria. If so, the event is discarded at step 615. If not, the event data is written to the database at step 616. The client at step 617 then checks to determine whether the bridge process has terminated. If so, the client terminates at step 618. Otherwise the client returns to steps 608 and 609 to await the receipt of new event data.

The above description explains how the event data is entered into the database 101. However, it is necessary to process the event data in order for the system to determine whether a coordinated intrusion attack is occurring. Presenting only the event data from the sensors to a user typically will overwhelm the user, and will result in false positive indications of an intrusion attack. As shown in Fig. 7, the invention provides a number of independent, modular correlators running in parallel, each of whose function is to receive the event data from the database and process it according to its own specific correlation rule to obtain various correlated event data. The correlated events as determined by the correlators are then written into the database by the correlators, from which the database provides the correlated event data to a user through the user interface or browser 105. The correlated event data is saved in the database along with the underlying sensor (raw) event data which resulted in the correlated event being determined.

As shown in Fig. 7, the event data correlators 104 are comprised of reactive correlators 701, regional

correlators 702, temporal correlators 703, IP correlators 704, and category correlators 705. Upon receipt of new event data, the database 101 sends the data to the correlators via a number of trigger/pipes. The correlators accept the event data, process it according to their particular correlation rule set, and upon determining the existence of a correlated event as a result of correlation processing, the correlators transmit the correlated event data to the database 101 for storage and for presentation to a user through the user interface.

The reactive correlator 701 interacts with tables provided in the database and sets a field in such tables which results in the data triggering a piece of code that changes the parameters of a sensor or other component outside the system. For example, the value set in the field may result in the closing of a session by a firewall (thus shutting out a hacker), reconfiguring the policy rules on a firewall (thus eliminating access to a system vulnerability), or causing an ISS, to scan the network for a vulnerability that a hacker is attempting to exploit.

The regional correlator 702 would receive data from the database and report the data to a central system which also receives data from regional correlators of a group of other systems through interactions with their respective databases. The central system would be provided with its own correlation capabilities to determine the existence of correlated events as a result of the reception of certain data or events from the multiple systems.

The temporal correlator 703 creates a correlated event for a specified sequence of signatures reported for a given pair of source and destination IPs, within a certain period of time or otherwise. The parameters for

the temporal correlator are a sequence of normalized signatures.

Fig. 8 illustrates the types of records stored in the database 101. Because sensor data from different sensors manufactured by different vendors typically are in formats which are not compatible with each other (in other words, use different terminology to describe the same type of occurrence), it is necessary to normalize the sensor data from the diverse sensors in order to meaningfully process and correlate the received event data. For this purpose, signature normalization look-up tables 801 are provided, in which vendor-specific event signatures are mapped to common signatures denoting the same type of event. For example, an ASIM 2.0 sensor may refer to a "spoofed IP" event or attack with a sensor signature "10012-X", while a NetRadar sensor may provide a signature of "10011-1" to denote this type of event. Further, NetRadar may provide a signature "10010-3" to refer to a port scan event, while a RealSecure sensor would provide a signature "ip-portscan" to denote this type of event. The signature normalization look-up table respectively maps each of these signatures to the respective common signatures "Spoofed IP" and "Port_Scan". Consequently, the system receiving these diverse signatures from the sensors will be able to determine that a common event has occurred by virtue of the normalization of the sensor signatures.

An event record 802 is written into the database for each new event parsed by the bridge from the data stream received from the sensor tap. This record includes information such as the event ID, the signature (sensor) name, the source IP address, and the destination IP address as received from each sensor tap. The correlation

category record 803 describes a predefined category of event. The normalized signature category record 804 maps a normalized signature to a category code. The event log record 805 associates correlated event IDs with event IDs. The event record 806 contains normalized event data.

Referring now to Fig. 9A, a flow chart of the operation of the category correlator 705 will be described. The category correlator retrieves raw sensor event data from the database and groups sensor signatures into a number of different categories. The number of signatures in each category is counted in a given time period. If the number of counted signatures in the time period in category x is greater than a predetermined number y, then the correlator will generate a correlated event of priority z.

After the correlator is started at step 901, a correlator initialization process is commenced at step 902, in which data structures and parameters used by the correlator are initialized. At step 903 the correlator reads from the database tables 803 and 804 (as shown in Fig. 8). Table 803 contains for each of the various defined categories, identified by a category code, the description of the category (x) and the threshold (y). The category code preferably is an integer, but may be any combination of alphanumeric characters. Table 804 contains for each category code a normalized sensor signature. A normalized signature can map to more than one category.

At step 904 the correlator creates a number of internal data structures, including a category array structure

category_array: struct cat_data {char description[MAX_CONSTANT1]; int threshold;}

which is indexed by category code; a normalized signature categories structure

normalized_signature_categories: struct sc {char sig_name[MAX_CONSTANT2]; linkedlist correlation_categories;}

which is a hash table keyed on normalized signatures; and a traffic hash table structure

traffic_hash-table: struct correlated_data { int category_code; int number_signatures; BOOLEAN is_correlated; char correlated_event_id[MAX_CONSTANT3]; LinkedList traffic_data;}

which is a hash table (initially empty) of categories for traffic sent from the database, or in other words a list of all events received.

At step 905, the correlator hashes on a normalized signature in the sensor event data read from the database and at step 906 determines whether the normalized signature exists in the normalized signature category table 804. If the normalized signature does not exist, an error condition exists and the process returns to check additional sensor event data. If the normalized signature is found in the table, for each category code in the linked list of normalized signatures as determined in step 907, the correlator hashes on the category code in the correlation category table 803 and compares the description with the sensor event data. If there is a

collision at step 909, the event is appended to the linked list of traffic data and the signature count is incremented at step 911. If there is no collision, at step 910 a new description is created and inserted into the traffic hash-table.

At step 912 it is then determined whether a correlated event already exists for this category. If so, at step 913 the event is appended to the already existing correlated event. If not, at step 914 the number of signatures in the traffic hash table is compared with the threshold given in the correlation category table 803. If the signature count exceeds the threshold, a correlated event is created at step 915 and written to the database. Otherwise, the process returns to check the next category code in the linked list, at step 907. The step 915 of creating a correlated event is comprised of the process shown in Fig. 9C. At step 9001, the next event_id is retrieved from the database. At step 9002, the correlated event data is stored in the event table of the database. At step 9003, the correlated_event_id and the is_correlated flag are set in the correlated_data field. At step 9004 the process determines whether additional items in the linked list of traffic data exist; if yes, the ordered pair of the correlated event ID and the event ID are stored in the event log; if no, the process exits.

The operation of the IP correlator 704 now will be described with reference to Figs. 9B-1 and 9B-2. The IP correlator counts signatures for each source IP address and each target IP address, and generates a correlated event if the number of signatures for any specific source IP x.x.x.x is greater than a predetermined threshold y, or

if the number of signatures for any specific target IP x.x.x.x is greater than a predetermined threshold y.

Initial processing for the IP correlator is substantially the same as shown in steps 901-904 of Fig. 9A. The IP correlator creates a traffic hash table data structure which contains the source IP addresses and target IP addresses for data traffic received from the database. As shown in Fig. 9B-1, after step 904, the IP correlator hashes on the source IP address in the sensor event data read from the database and at step 917 determines whether there is a collision. If not, at step 918 correlated data is created and inserted for this IP. If yes, at step 919 it is determined whether a source for the source IP is found. If yes, at step 920 the source IP event is appended to the linked list for the traffic data, and the signature count for this source IP signature is incremented. If not, the process goes to step 918 as explained above. The process then proceeds to step 921, where it is determined if a correlated event already exists. If so, at step 922 the event is appended to the already existing correlated event. If not, at step 923 the number of signatures in the traffic hash table is compared with the threshold given in the correlation category table 803. If the signature count exceeds the threshold, a correlated event is created at step 924 and written to the database. Otherwise, the process returns to check the target IP in the linked list, at step 925.

Referring to Fig. 9B-2, at step 926 the IP correlator hashes on the target IP address in the sensor event data read from the database and at step 927 determines whether there is a collision. If not, at step 928 correlated data is created and inserted for this IP. If yes, at step 929

it is determined whether a source for the target IP is found. If yes, at step 930 the target IP event is appended to the linked list for the traffic data, and the signature count for this target IP signature is incremented. If not, the process goes to step 928 as explained above. The process then proceeds to step 931, where it is determined if a correlated event already exists. If so, at step 932 the event is appended to the already existing correlated event. If not, at step 933 the number of signatures in the traffic hash table is compared with the threshold given in the correlation category table 803. If the signature count exceeds the threshold, a correlated event is created at step 934 and written to the database. Otherwise, the process returns to step 916 to check the next source IP in the linked list, at step 935. The steps 924, 934 of creating correlated events each comprise the same steps as shown in Fig. 9C.

The invention having been thus described, it will be apparent to those skilled in the art that the same may be varied in many ways without departing from the spirit and scope of the invention. For example, while processes have been disclosed for category correlators and IP correlators, many other types of correlations may be performed to generate correlated events, such as boolean correlators or correlators which key on specific sequences of sensor signatures.

WHAT IS CLAIMED IS:

1. A system for detecting attempted intrusions into a computer network composed of a plurality of computer hosts, comprising:

a central intrusion detection database system;

a tap residing on at least one computer host, which reads event data from activity log files created by an event sensor provided on the host and sends the event data to said central database system;

said central database system including for said tap a bridge which receives and processes the sensor information from the tap, and stores the processed sensor information in a database;

at least one correlator which receives the processed sensor information from the database, processes the information according to a predefined algorithm to generate a correlated event, and stores the correlated event in the database; and

a user interface for receiving the information stored in the database and presenting the received information to a user for evaluation.

2. A system according to claim 1, wherein said tap creates a TCP server in the host on which said tap resides, and said bridge creates a TCP client in the central database system, for communicating with said TCP server over a TCP socket established over said network.

3. A system according to claim 2, wherein said bridge is located within a boundary protected by a boundary device, and said TCP client initiates communication with said TCP server through said boundary device.

4. A system according to claim 1, wherein said correlator correlates events identified by specific sensor output signature data.

5. A system according to claim 1, wherein said correlator correlates events identified by specific IP network address data.

6. A system according to claim 1, wherein said tap automatically sends new event data to said bridge as the new event data becomes available on the host.

7. A system according to claim 1, wherein said tap parses the event data from said activity log files into output data strings.

8. A system according to claim 7, wherein said bridge parses said output data strings into specific variable categories for storing in said database.

9. A system for detecting attempted intrusions into a computer network composed of a plurality of computer hosts, comprising:

a central intrusion detection database system;

a plurality of taps each residing on a corresponding computer host, which taps each read event data from activity log files created by an event sensor provided on its respective host and sends the event data to said central database system;

said central database system including for each tap a bridge which receives and processes the sensor information

from the tap, and stores the processed sensor information in a database;

a plurality of correlator modules, each running in parallel on said central database system, each of which receives the processed sensor information from the database, processes the information according to a predefined algorithm to generate a correlated event, and stores the correlated event in the database; and

a user interface for receiving the information stored in the database and presenting the received information to a user for evaluation.

10. A system according to claim 9, wherein each of said taps creates a TCP server in the host on which the tap resides, and each said bridge creates a TCP client in the central database system, for communicating with its corresponding TCP server over a TCP socket established over said network.

11. A system according to claim 20, wherein at least one bridge is located within a boundary protected by a boundary device, and the TCP client created by said at least one bridge initiates communication with TCP server of the bridge's corresponding tap through said boundary device.

12. A system according to claim 9, wherein one of said correlators correlates events identified by specific sensor output signature data.

13. A system according to claim 9, wherein one of said correlators correlates events identified by specific IP network address data.

14. A system according to claim 9, wherein said taps automatically send new event data to said bridges as the new event data becomes available on their respective hosts.

15. A system according to claim 9, wherein said taps parse the event data from said activity log files into output data strings.

16. A system according to claim 15, wherein said bridges parse said output data strings into specific variable categories for storing in said database.

17. A method for detection of concerted intrusion efforts on a computer network, comprising the steps of:

collecting data from multiple local sensors resident in intrusion detection system log files, firewalls, or routers across multiple sites on said network;

normalizing the sensor data from said multiple different sensors resident on the multiple network sites and storing said normalized data in a database;

processing the normalized data in a plurality of correlator modules each running in parallel and each operating according to a different correlation algorithm;

generating in said correlators correlated events when correlation of said normalized data exceeds a predetermined threshold;

storing said correlated events in said database in association with said sensor data; and

presenting said correlated events and associated sensor data to a user through a user interface.

US01 22624

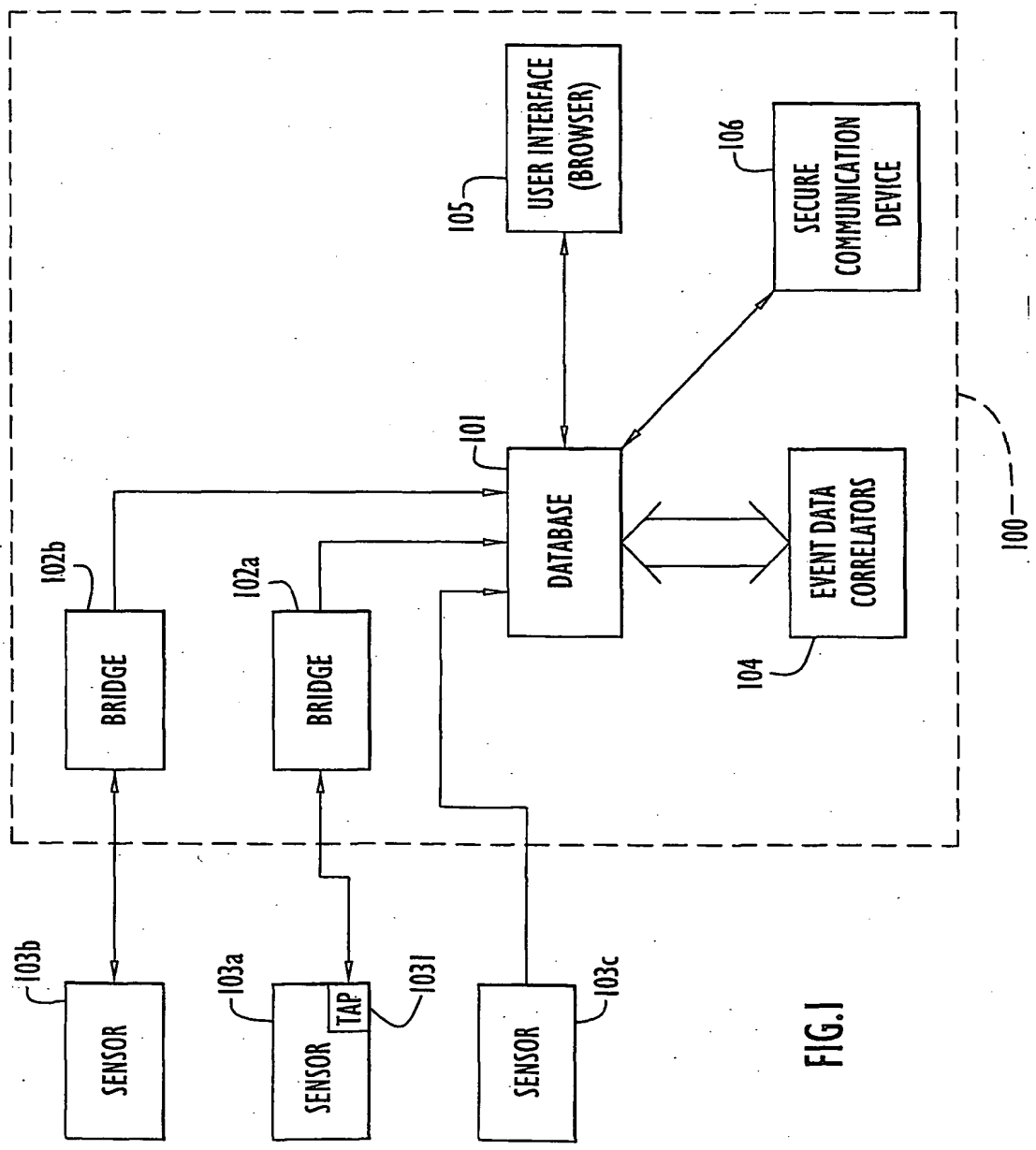


FIG.1

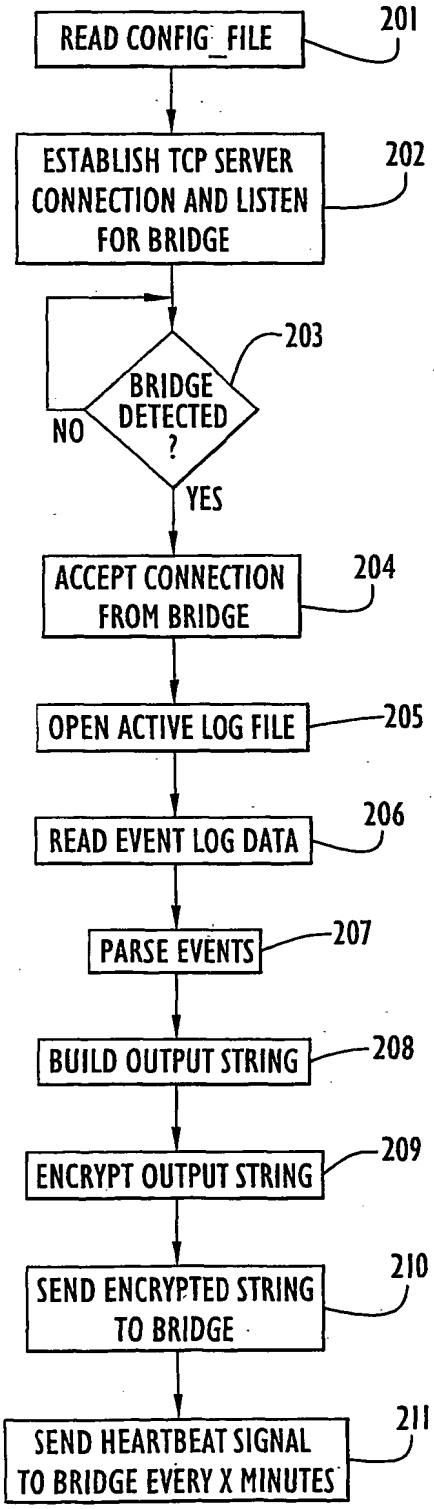

READ CONFIG_FILE
201
ESTABLISH TCP SERVER CONNECTION AND LISTEN FOR BRIDGE
202
BRIDGE DETECTED ?
203
NO
YES
ACCEPT CONNECTION FROM BRIDGE
204
OPEN ACTIVE LOG FILE
205
READ EVENT LOG DATA
206
PARSE EVENTS
207
BUILD OUTPUT STRING
208
ENCRYPT OUTPUT STRING
209
SEND ENCRYPTED STRING TO BRIDGE
210
SEND HEARTBEAT SIGNAL TO BRIDGE EVERY X MINUTES
211

FIG.2

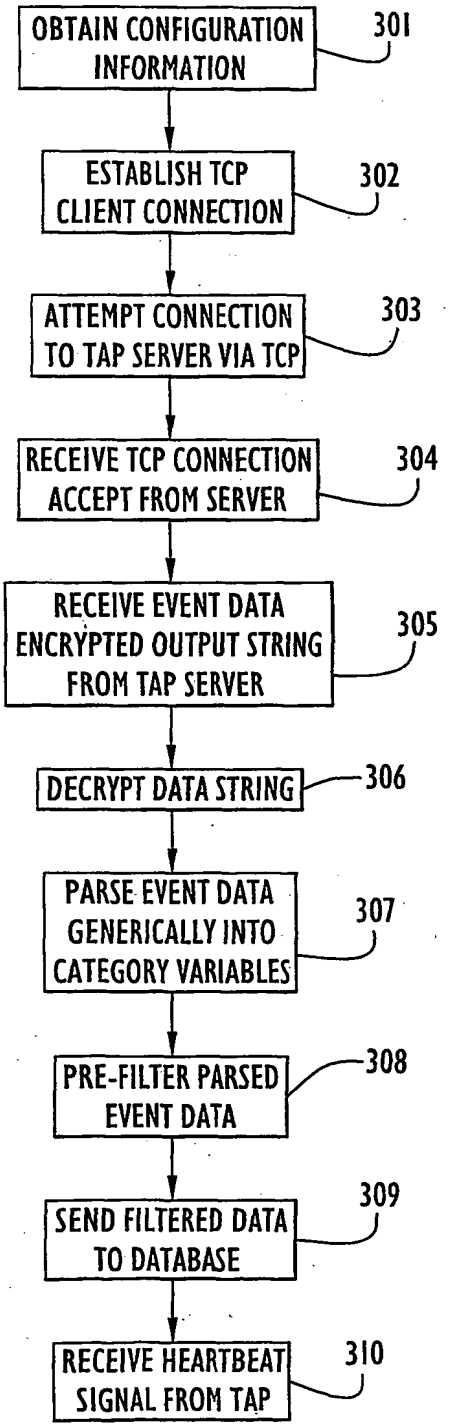

OBTAIN CONFIGURATION INFORMATION
301
ESTABLISH TCP CLIENT CONNECTION
302
ATTEMPT CONNECTION TO TAP SERVER VIA TCP
303
RECEIVE TCP CONNECTION ACCEPT FROM SERVER
304
RECEIVE EVENT DATA ENCRYPTED OUTPUT STRING FROM TAP SERVER
305
DECRYPT DATA STRING
306
PARSE EVENT DATA GENERICALLY INTO CATEGORY VARIABLES
307
PRE-FILTER PARSED EVENT DATA
308
SEND FILTERED DATA TO DATABASE
309
RECEIVE HEARTBEAT SIGNAL FROM TAP
310

FIG.3

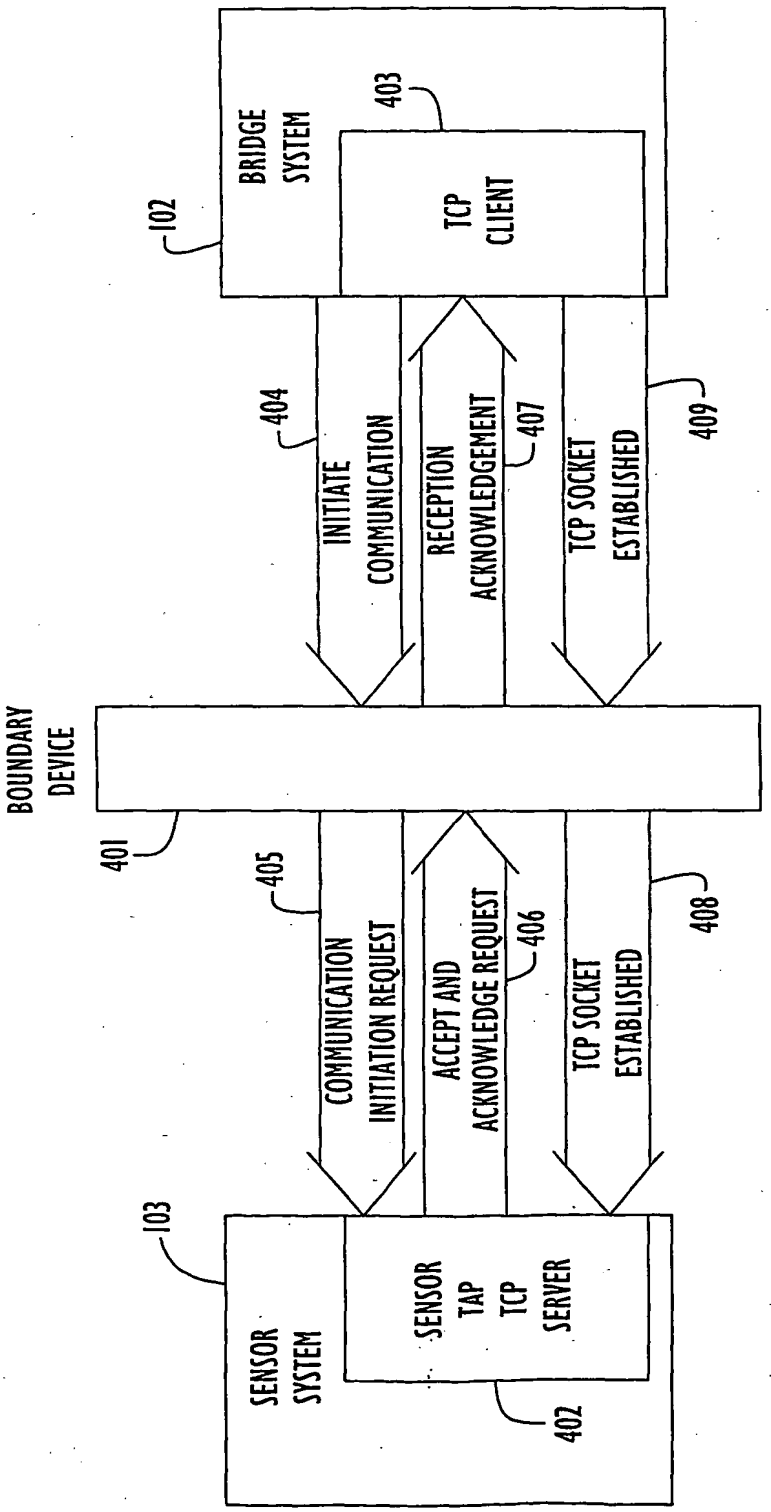

BRIDGE SYSTEM
102
TCP CLIENT
403
INITIATE COMMUNICATION
404
RECEPTION ACKNOWLEDGEMENT
407
TCP SOCKET ESTABLISHED
409
INSIDE
BOUNDARY DEVICE
401
COMMUNICATION INITIATION REQUEST
405
ACCEPT AND ACKNOWLEDGE REQUEST
406
TCP SOCKET ESTABLISHED
408
OUTSIDE
SENSOR SYSTEM
103
SENSOR TAP TCP SERVER
402

FIG.4

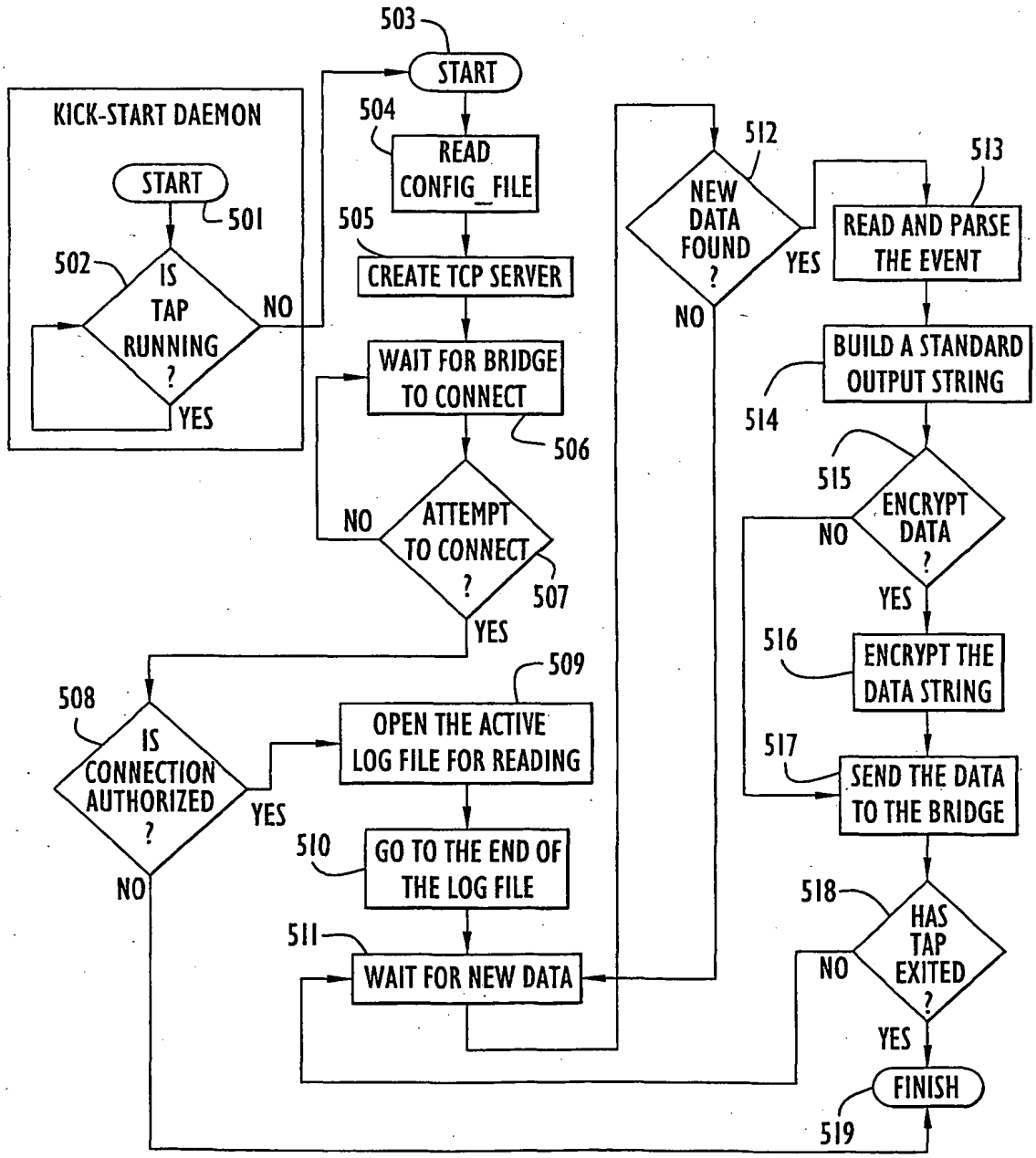

KICK-START DAEMON
START
501
IS TAP RUNNING ?
502
NO
YES
503
START
504
READ CONFIG_FILE
505
CREATE TCP SERVER
WAIT FOR BRIDGE TO CONNECT
506
ATTEMPT TO CONNECT ?
507
NO
YES
IS CONNECTION AUTHORIZED ?
508
YES
NO
509
OPEN THE ACTIVE LOG FILE FOR READING
510
GO TO THE END OF THE LOG FILE
511
WAIT FOR NEW DATA
NEW DATA FOUND ?
512
YES
NO
513
READ AND PARSE THE EVENT
BUILD A STANDARD OUTPUT STRING
514
ENCRYPT DATA ?
515
NO
YES
ENCRYPT THE DATA STRING
516
SEND THE DATA TO THE BRIDGE
517
HAS TAP EXITED ?
518
NO
YES
FINISH
519

FIG.5

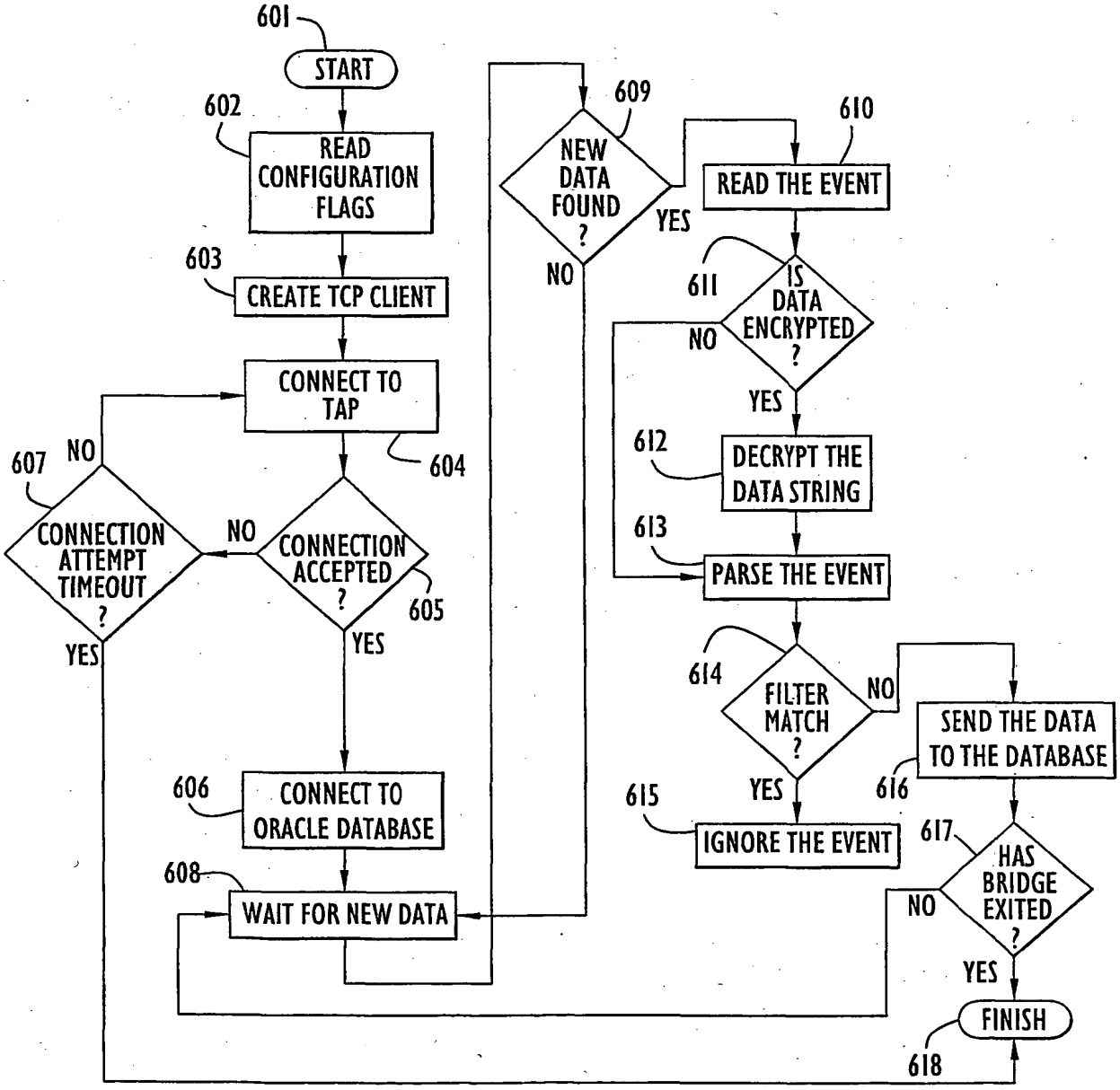

601
START
602
READ CONFIGURATION FLAGS
603
CREATE TCP CLIENT
CONNECT TO TAP
604
607
NO
CONNECTION ATTEMPT TIMEOUT ?
YES
NO
CONNECTION ACCEPTED ?
605
YES
606
CONNECT TO ORACLE DATABASE
608
WAIT FOR NEW DATA
609
NEW DATA FOUND ?
YES
NO
610
READ THE EVENT
611
IS DATA ENCRYPTED ?
NO
YES
612
DECRYPT THE DATA STRING
613
PARSE THE EVENT
614
FILTER MATCH ?
NO
YES
615
IGNORE THE EVENT
SEND THE DATA TO THE DATABASE
616
617
HAS BRIDGE EXITED ?
NO
YES
FINISH
618

FIG.6

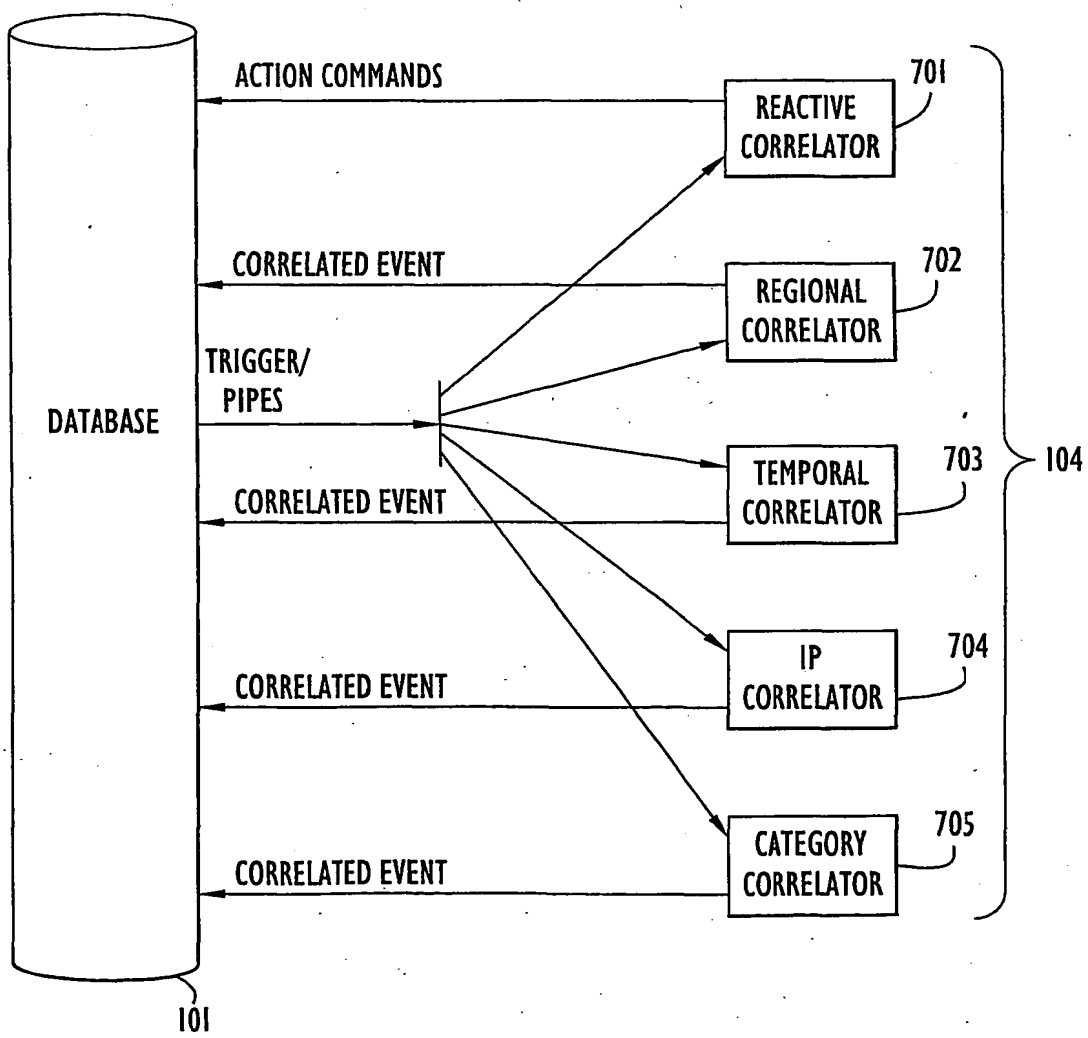

ACTION COMMANDS
CORRELATED EVENT
TRIGGER/
PIPES
CORRELATED EVENT
CORRELATED EVENT
CORRELATED EVENT
DATABASE
101
REACTIVE
CORRELATOR
701
REGIONAL
CORRELATOR
702
TEMPORAL
CORRELATOR
703
IP
CORRELATOR
704
CATEGORY
CORRELATOR
705
104

FIG.7

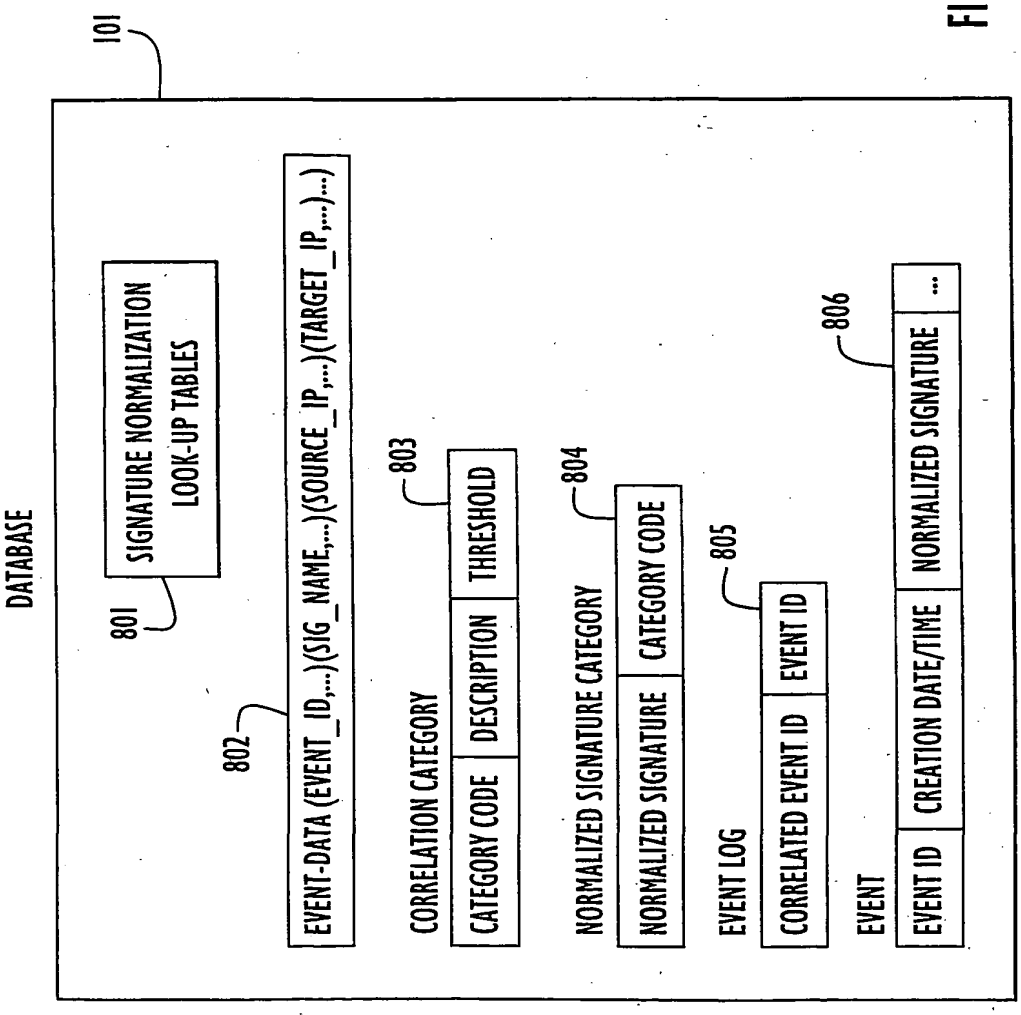

101
DATABASE
801
SIGNATURE NORMALIZATION LOOK-UP TABLES
802
EVENT-DATA (EVENT_ID,...)(SIG_NAME,...)(SOURCE_IP,...)(TARGET_IP,...)...)
803
CORRELATION CATEGORY
CATEGORY CODE
DESCRIPTION
THRESHOLD
804
NORMALIZED SIGNATURE CATEGORY
NORMALIZED SIGNATURE
CATEGORY CODE
805
EVENT LOG
CORRELATED EVENT ID
EVENT ID
806
EVENT
EVENT ID
CREATION DATE/TIME
NORMALIZED SIGNATURE
...

FIG.8

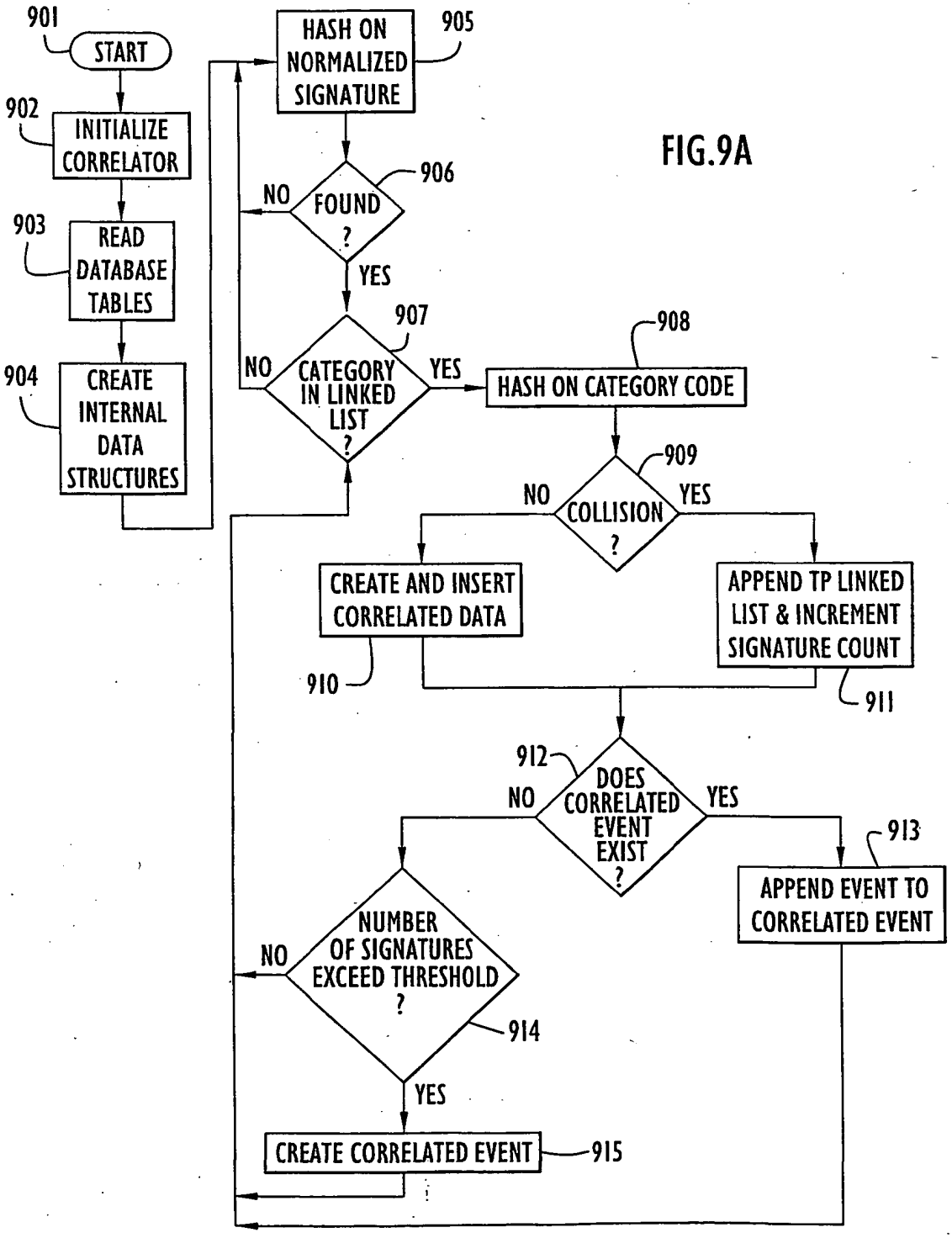

901
START
902
INITIALIZE CORRELATOR
903
READ DATABASE TABLES
904
CREATE INTERNAL DATA STRUCTURES
905
HASH ON NORMALIZED SIGNATURE
906
FOUND ?
NO
YES
907
CATEGORY IN LINKED LIST ?
NO
YES
908
HASH ON CATEGORY CODE
909
COLLISION ?
NO
YES
CREATE AND INSERT CORRELATED DATA
910
APPEND TP LINKED LIST & INCREMENT SIGNATURE COUNT
911
912
DOES CORRELATED EVENT EXIST ?
NO
YES
913
APPEND EVENT TO CORRELATED EVENT
NUMBER OF SIGNATURES EXCEED THRESHOLD ?
NO
914
YES
CREATE CORRELATED EVENT
915

FIG.9A

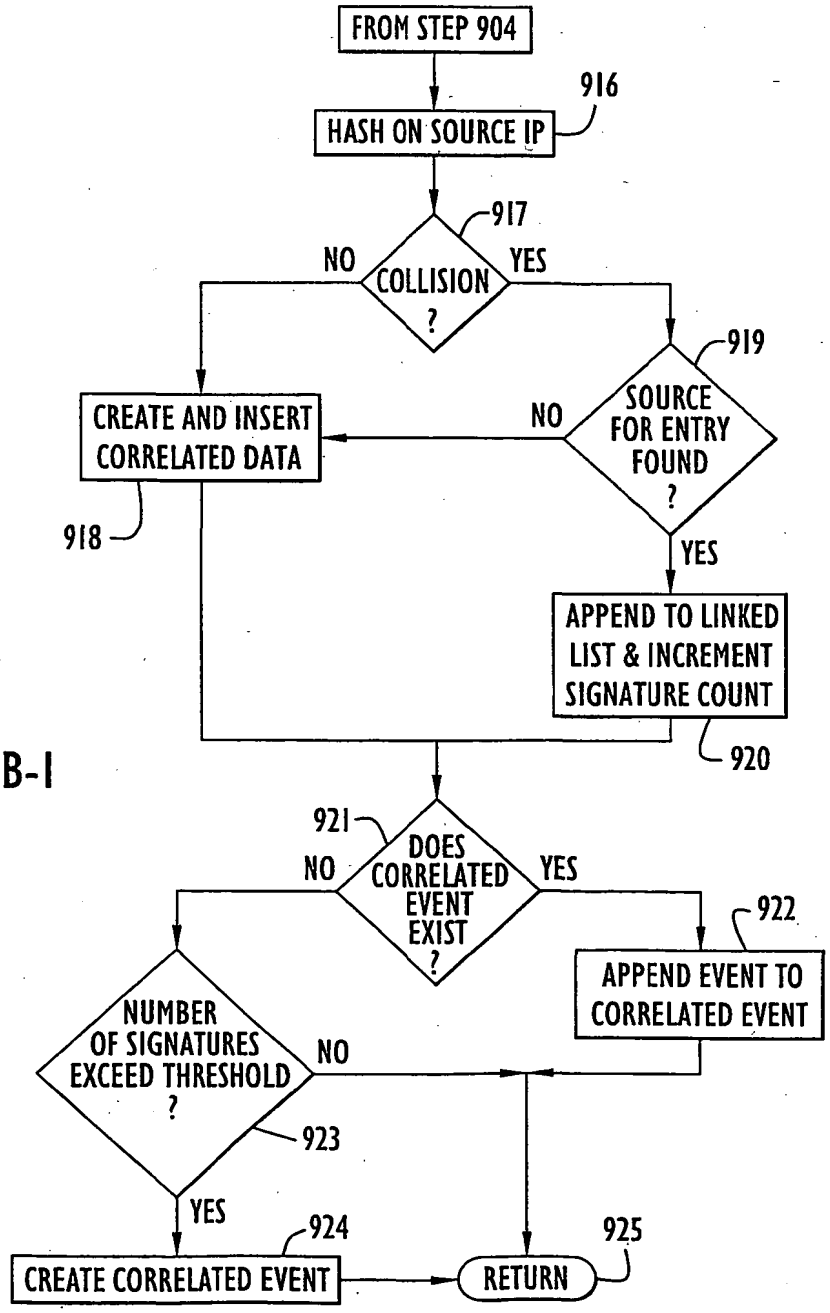

FROM STEP 904
916
HASH ON SOURCE IP
917
NO
COLLISION ?
YES
919
SOURCE FOR ENTRY FOUND ?
NO
CREATE AND INSERT CORRELATED DATA
918
YES
APPEND TO LINKED LIST & INCREMENT SIGNATURE COUNT
920
921
DOES CORRELATED EVENT EXIST ?
NO
YES
922
APPEND EVENT TO CORRELATED EVENT
NUMBER OF SIGNATURES EXCEED THRESHOLD ?
NO
923
YES
924
CREATE CORRELATED EVENT
925
RETURN

FIG.9B-1

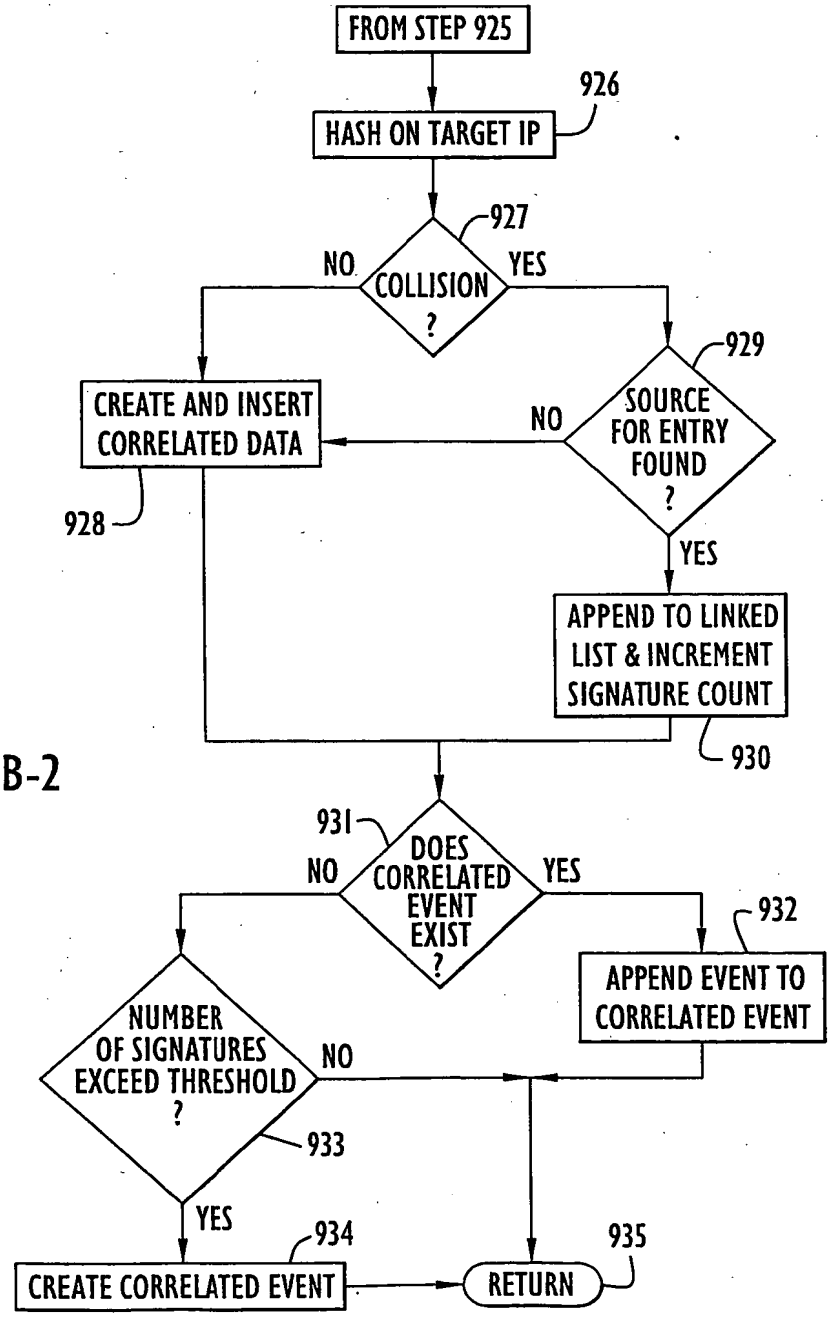

FROM STEP 925
926
HASH ON TARGET IP
927
NO
COLLISION ?
YES
929
CREATE AND INSERT CORRELATED DATA
NO
SOURCE FOR ENTRY FOUND ?
928
YES
APPEND TO LINKED LIST & INCREMENT SIGNATURE COUNT
930
931
NO
DOES CORRELATED EVENT EXIST ?
YES
932
APPEND EVENT TO CORRELATED EVENT
NUMBER OF SIGNATURES EXCEED THRESHOLD ?
NO
933
YES
934
CREATE CORRELATED EVENT
935
RETURN

FIG.9B-2

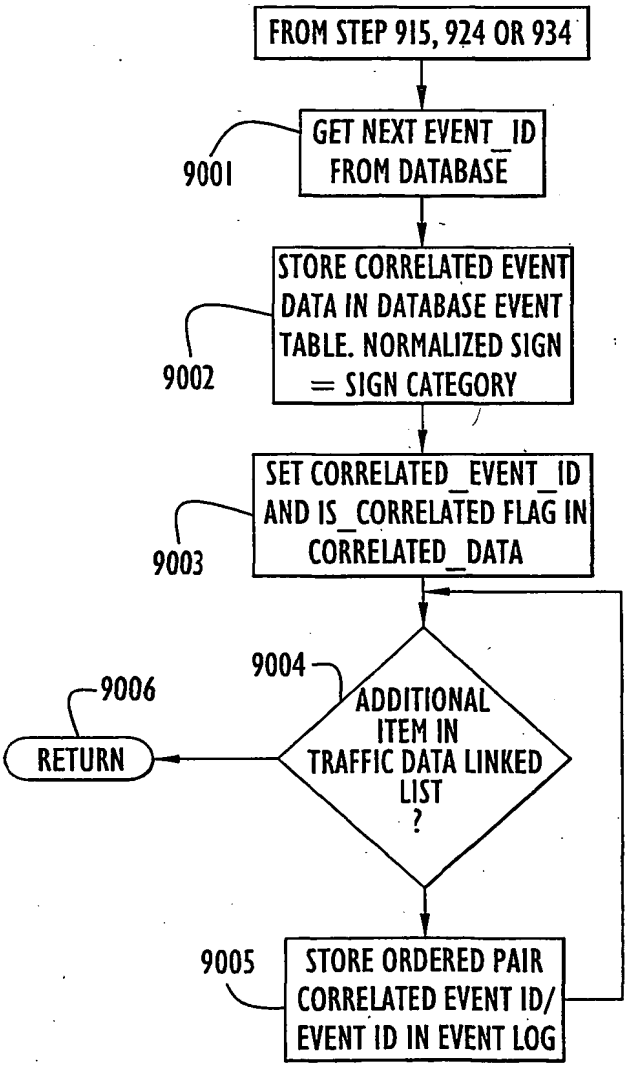

FROM STEP 915, 924 OR 934
9001
GET NEXT EVENT_ID FROM DATABASE
9002
STORE CORRELATED EVENT DATA IN DATABASE EVENT TABLE. NORMALIZED SIGN = SIGN CATEGORY
9003
SET CORRELATED_EVENT_ID AND IS_CORRELATED FLAG IN CORRELATED_DATA
9004
ADDITIONAL ITEM IN TRAFFIC DATA LINKED LIST ?
9006
RETURN
9005
STORE ORDERED PAIR CORRELATED EVENT ID/ EVENT ID IN EVENT LOG

FIG.9C